# Math 438 Optimal Control Project

Tyler Mansfield, Eric Todd

April 10, 2020

**Abstract**

Optimal path planning is a problem that has been solved in countless mathematical frameworks and models. We apply optimal control theory, specifically Pontryagin's maximum principle, to formulate paths that reach a desired destination and avoid various moving obstacles in a minimum amount of time. We discuss the mathematical framework, solution derivation, and numerical methods used to obtain our solution. Several examples demonstrate the results of our methods as well as advantages and disadvantages to our particular model.

## 1 Introduction

The task of finding optimal paths has been a relevant problem since the beginnings of the automated robot. A common constraint on the search of optimal paths includes the task of avoiding obstacles. This type of problem is a subset of what are known as kinodynamic planning problems where the goal is to control robot motion subject to constraints such as fixed beginning and end states, avoiding obstacles (moving or not), as well as bounds on velocity and acceleration. Many different approaches and techniques have been applied to this general class of problems, including dynamic programming, the bellman optimality principle, state-space models, partial differential equations, and manipulation of geometric structures. [1]

Even with stationary obstacles, this type of optimal path finding has been a historically challenging task. As an example, finding the shortest path for a point "robot" in three dimensions with polyhedral obstacles is known to be a NP (nondeterministic polynomial) hard problem. [2] Thus, efficient algorithms are often needed to seek for pseudo-optimal solutions since explicit forms of the solution are rarely available.

We seek to solve the problem of kinodynamic motion planning in $\mathbb{R}^2$ with constant-velocity obstacles using the framework of this class (optimal control). Our results can easily be generalized to more general obstacles of arbitrary shapes and velocity, but our specific implementation assumes that the obstacle path is known beforehand.

## 2 Mathematical Description

Suppose we are trying to get from point A to point B in $\mathbb{R}^2$ in the shortest time possible. Without loss of generality, we can assume that A is located at the origin and B is the point $(d, 0)$, where $d$ is the distance between A and B. Suppose our state vector is given by:

$$\vec{s}(t) = \begin{bmatrix} x(t) \\ y(t) \\ x'(t) \\ y'(t) \end{bmatrix}$$

The components of our state vector $\vec{s}$ include $(x(t), y(t))$, which is our position at time t in the Cartesian plane, and $(x'(t), y'(t))$, which is our velocity vector at time t. From here forward, we will refer to elements of our state vector by $s_1(t), s_2(t)$,... etc rather than $x(t), y(t)$, ... etc for consistency.

Thus our problem amounts to minimizing the functional

$$J(\vec{u}) = \int_{t_0}^{t_f} 1 dt$$

where we require that we start and end with zero velocity in addition to meeting the positional requirements of starting at point A and ending at point B. In our state representation these conditions can be written as:

$$\vec{s}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \vec{s}(t_f) = \begin{bmatrix} d \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

Our control for this problem will be the acceleration of our particle, with $u_1(t)$ being our acceleration in the x-direction and $u_2(t)$ the acceleration in the y-direction. Thus we have the following relationship:

$$\dot{\vec{s}}(t) = \begin{bmatrix} s_1'(t) \\ s_2'(t) \\ s_3'(t) \\ s_4'(t) \end{bmatrix} = \begin{bmatrix} s_3(t) \\ s_4(t) \\ u_1(t) \\ u_2(t) \end{bmatrix} \tag{2}$$

with some restraints or penalties associated with large values of $\vec{u}(t)$ to model acceleration bounds (to be discussed more upcoming).

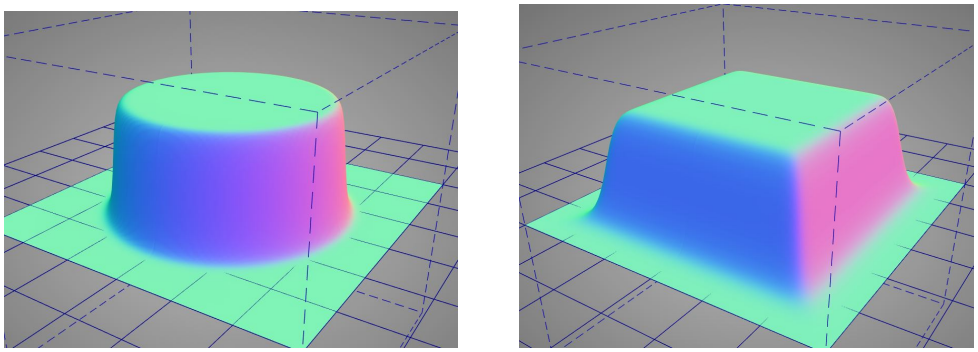
Now suppose we have $n$ moving obstacles in our path that we must avoid. For simplicity, each obstacle is either (i) a rectangle with dimensions $l_x$ by $l_y$ or (ii) an ellipse with radius of length $r_x$ in the x-direction and length $r_y$ in the y-direction. The $i^{th}$ obstacle has a starting initial center of $(x_{0_i}, y_{0_i})$ and constant velocity of $(v_{x_i}, v_{y_i})$. We can modify our functional to avoid hitting these obstacles by creating a new cost function to minimize:

$$J(\vec{u}) = \int_{t_0}^{t_f} (1 + \sum_{i=1}^{n} C_i(\vec{s}, t))dt \tag{3}$$

subject to conditions (1) and (2) where

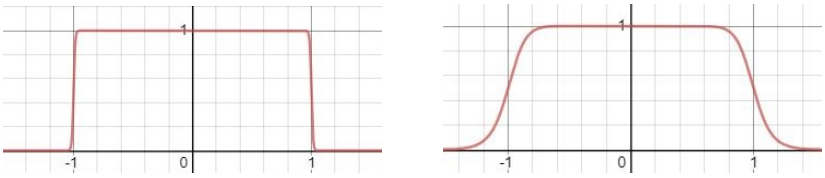
$$C_i(\vec{s}, t) = \begin{cases} \frac{K}{(\frac{1}{r_x}(s_1 - v_{x_i}t - x_{0_i})^2 + \frac{1}{r_y}(s_2 - v_{y_i}t - y_{0_i})^2)^\lambda + 1} & \text{if } C_i \text{ is ellipse} \\ \frac{K}{(\frac{2}{l_x}(s_1 - v_{x_i}t - x_{0_i}))^\lambda + (\frac{2}{l_y}(s_2 - v_{y_i}t - y_{0_i}))^\lambda + 1} & \text{if } C_i \text{ is rectangle} \end{cases} \tag{4}$$

In the above formulation, $K > 0$ is a scaling parameter that creates larger penalties for intersecting the obstacle as $K \to \infty$. Our other parameter, $\lambda \in \mathbb{Z}^+$ is a natural number that "softens" the boundary for our obstacle as $\lambda \to 0$. In other words, for large $\lambda$ the cost function is penalized when the path intersects the obstacle, but is not affected otherwise. Small values of $\lambda$ correspond to a softer boundary that begins to penalize the path as it approaches the obstacle (see Figure 2). Note that $\lambda$ must be even for rectangular obstacles in order to achieve the desired result.



(a) Ellipse Cost Function ($\lambda = 20$) (b) Rectangle Cost Function ($\lambda = 20$)

Figure 1: Visualization of our Cost Functions from Equation (4). [3]




(a) Large $\lambda$ (b) Small $\lambda$

Figure 2: Obstacle penalties for varying $\lambda$ values. The boundary of the object is at x = -1, x = 1. [4]

Finally, rather than bounding our acceleration (such as requiring $u_1^2(t) + u_2^2(t) \leq 1$) it may be useful to pose this problem into an unconstrained optimization problem to make it easier to solve. To that end, we propose modifying our functional to be:

$$J(\vec{u}) = \int_{t_0}^{t_f} \big(1 + \sum_{i=1}^{n} C_i(\vec{s}, t) + M(u_1^2(t) + u_2^2(t))\big)dt \tag{5}$$

where M is a constant that weights the penalty for large acceleration.

# 3 Method Description

We plan to use Pontryagin's Maximum Principle to solve this problem. In our system, the control is the acceleration, which is similar to the double integrator problem discussed in the textbook. However, our functional, J, is much more complicated since it involves penalizing the particle for colliding with any of the moving obstacles as well as large acceleration values.

The Maximum Principle allows us to set up a system of 8 ODEs posed as a Boundary Value problem that can solved with numerical methods, such as Python's scipy.integrate.solve_bvp. Though this method requires some analytical work ahead of time to formulate the equations and a good initial guess, we often find successful results.

# 4 Solution

Given our functional (5) and governing ODE system (2), our Hamiltonian can be written as:

$$H(t, \vec{p}, \vec{s}, \vec{u}) = p_1 s_3 + p_2 s_4 + p_3 u_1 + p_4 u_2 - L(t, \vec{s}, \vec{u}) \tag{6}$$

where $L(t, \vec{s}, \vec{u})$ is our Lagrangian from the functional. By the Maximum Principle, our system of differential equations for our optimal costate vector $\vec{p^*}$ can be written as:

$$\dot{p_1^*} = -H_{s_1} = \sum_{i=1}^{n} \frac{d}{ds_1} C_i(\vec{s}, t) \tag{7}$$

$$\dot{p_2^*} = -H_{s_2} = \sum_{i=1}^{n} \frac{d}{ds_2} C_i(\vec{s}, t) \tag{8}$$

$$\dot{p_3^*} = -H_{s_3} = -p_1 \tag{9}$$

$$\dot{p_4^*} = -H_{s_4} = -p_2 \tag{10}$$

We also know that $H_{u_1^*} = H_{u_2^*} = 0$ by the Maximum Principle. This gives us that for our optimal acceleration we have $u_1^*(t) = \frac{1}{2M} p_3^*(t)$ and $u_2^*(t) = \frac{1}{2M} p_4^*(t)$. By making this substitution for u into (2), we now have a system of eight ODE's (four for the state and four for the costate) as well as the eight boundary conditions given by (1). This gives us an adequate set-up that we can pass into the function scipy.integrate.solve_bvp.

We still have an unknown parameter $t_f$ that we have yet to solve for. Values of $t_f$ that are too small will require large acceleration to reach our final destination and will penalize our functional $J$. In addition, values of $t_f$ that are too large will cause the constant term (the running cost) of our functional to integrate to a large value. Thus we propose the following algorithm to approximate the optimal value of $t_f$:

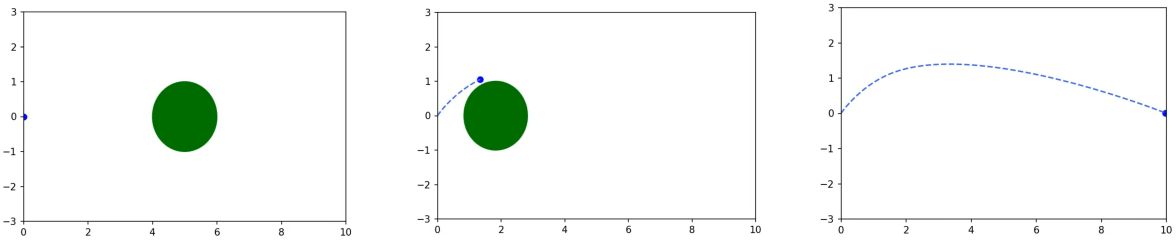
1. Make a guess for $t_f$.
2. Create a list of 40 values between $0.5 * t_f$ and $1.5 * t_f$.
3. Solve the optimal path problem for each value of $t_f$ in our list.
4. Evaluate the optimal paths in our functional and select the value of $t_f$ that gives the lowest total cost.
5. If the optimal $t_f$ chosen above is one of the 3 lowest or 3 highest values in our list, perhaps our initial guess was off. Set the optimal value of $t_f$ found above to be our new guess and repeat steps (2-5) until our optimal value for $t_f$ is not one of the 3 lowest or 3 highest values in our list.

# 5 Interpretation

Overall, this method worked very well on several examples that we tested. However, our solution depends heavily on our initial guess for the optimal path, as well as hyperparameter values of $K_i$, $\lambda_i$ and $M$, which scale the penalties for intersecting obstacle $i$, "sharpness" of the loss function for obstacle $i$, and large acceleration values respectively. We discuss our results on a few example problems, as well as some of the strengths and limitations of our method.

## 5.1 Example 1: Object Approaching the Origin

Our first example has the following set up: Assume we are trying to move from the point $(0, 0)$ to $(10, 0)$, with one moving circle in our path. The obstacle has radius 1, and begins centered at $(5, 0)$ and moves in a straight path towards the origin at a constant speed of $v_x = -1$. We set $K_1 = 50$, $M = 5$, and $\lambda_1 = 20$. The initial set up can be seen in Figure 3(a).



(a) Initial Setup (b) Avoiding the Obstacle (c) Full Solution Path

Figure 3: Our Method's Solution for Example 1

Our method was able to find a control on the acceleration of the particle that minimized the cost of the functional. The path resulting from applying this optimal control avoids the obstacle and reaches the desired target point $(10, 0)$ in near optimal time. It is interesting to note that the optimal control chooses to narrowly miss the obstacle as seen in Figure 3(b), in order to quickly readjust its trajectory towards the target point. This allows the running cost part of our functional (see equation 5) to be minimized while also avoiding the penalty of hitting the obstacle.

The naive "solution" to this problem would be to ignore all the obstacles and travels in a straight line to the target using an acceleration that would be optimal if their were no obstacles present. When plugging this naive control into our functional, the solution had a total cost of total cost of 68.146. Since our method avoided the obstacle, its total cost is much less, with a total of 17.012. A breakdown of the cost over time for our solution is seen in Figure 4. We can see that even though our method did not hit the obstacle, it was penalized a little for getting so close. The majority of the cost came from the acceleration and running cost of time.

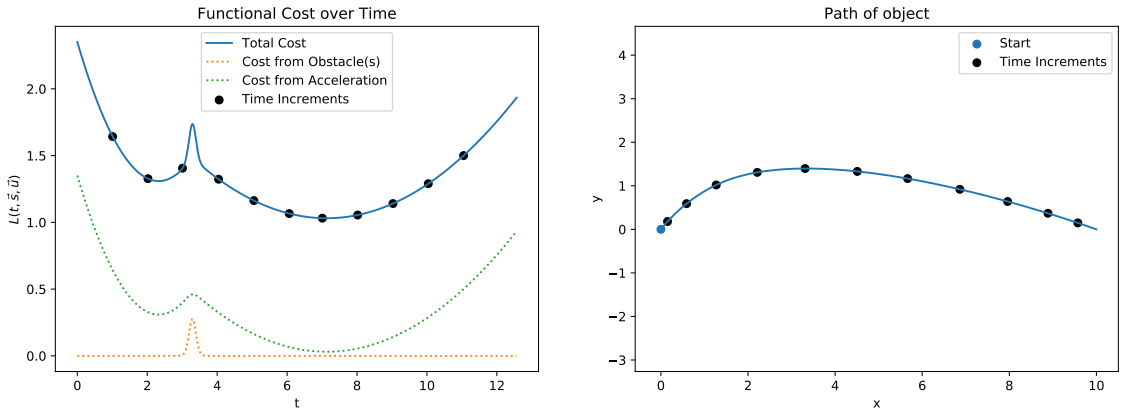
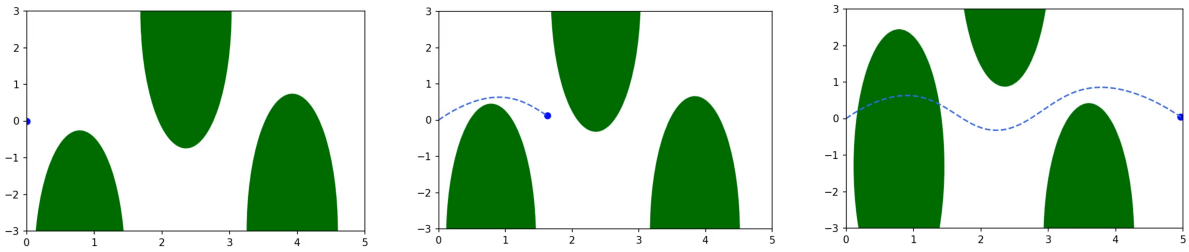



Figure 4: Example 1 - Solution Cost Analysis

## 5.2 Example 2: Twists and Turns

Our second example is a little more complicated than the first. We introduced 3 moving ellipse obstacles that were larger than the circle in Example 1. The first and second ellipses moved vertically upwards, while the third moved to the downward and to the left. Our goal this time was to move from the point $(0, 0)$ to $(5, 0)$ while avoiding the moving ellipses in our path. We set $K_i = 160$ and $\lambda_i = 26$ for each obstacle and $M = 4$. The initial set up for this example can be seen in Figure 5(a).



(a) Initial Setup (b) Avoiding the $1^{st}$ Obstacle (c) Full Solution Path

Figure 5: Our Method's Solution for Example 2

We can see in Figure 5(c) that our solution's path intersects with the path of the first obstacle, but the intersection occurs after the particle has already passed the obstacle on its way to the target. We see a similar strategy of avoiding the edges of obstacles just enough so that the control can readjust the particle's trajectory to continue towards the target point. The introduction of multiple obstacles increased the cost of our solution due to acceleration required to alter the trajectory repeatedly and for the path's proximity to the obstacles. However, we are pleased that our method's solution was able to still avoid all of the obstacles, and used a different approach than in Example 1. A breakdown of the cost over time for our solution can be seen in Figure 6.

As in Example 1, the naive solution to this problem ignores all the obstacles and travels in a straight line to the target. The naive solution had a total cost of total cost of 389.168. This is expected since there were three obstacles, each with a penalty of $K_i = 160$. The solution our method produced had a total cost of 22.674. Since our method was able to avoid the obstacles, its cost is much less than the naive method, but was more expensive than in Example 1. This is partly because the control had to change its trajectory more often than before, but also because the cost from getting close to the obstacles without hitting them increased with the number of obstacles the path had to avoid (see Figure 6).

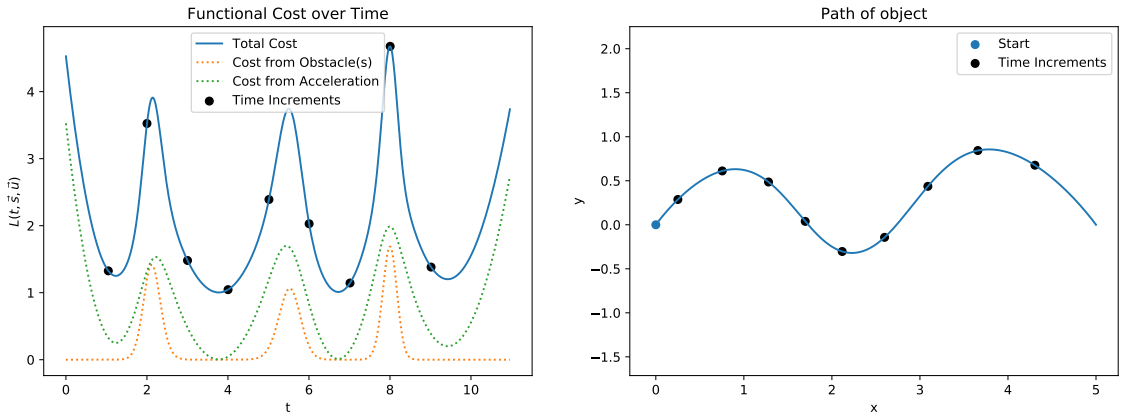



Figure 6: Example 2 - Solution Cost Analysis

## 5.3 Choosing Model Parameters

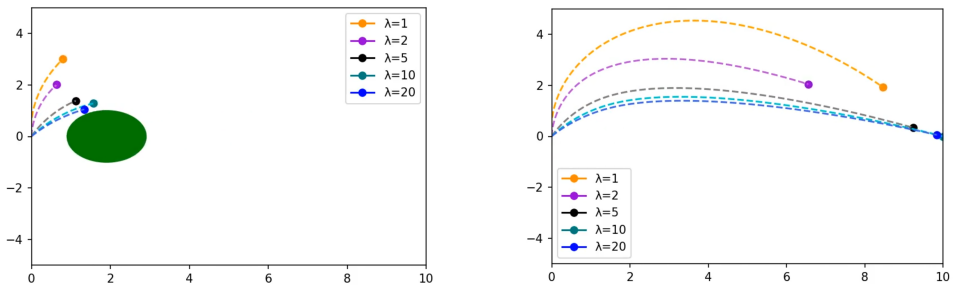
One advantage of our model is that we can choose the value of $M$ in our functional (5), and the values of $\lambda_i$ and $K_i$ for each obstacle. Since these hyperparameters penalize various features of a proposed path, they give us extra control when trying to model different physical phenomena. For example, modeling a car's optimal path and obstacles will be different than a UAV, or a ship. We discuss below the effects that being able to change $K$, $\lambda$ and $M$ have on our model.

### 5.3.1 How $K$ Influences our Model

The value of $K_i$ is roughly the penalty that each obstacle $C_i$ gives when our path intersects with it. We noticed if we made $K_i$ too small, our solution would choose to go through the obstacle rather than avoid it, because the penalty was not as large as the extra time it would take to go around it. We also saw that if $K_i$ was very large, the particle would make less intuitive choices to avoid the obstacle. This included staying still until a certain obstacle passed or even moving backwards to take a much longer path around the obstacle to the target. Thus, the value of $K_i$ that we choose allows us to influence our model's desire to avoid specific obstacles, and what kinds of paths it will consider.

### 5.3.2 How $\lambda$ Influences our Model

In this section, we analyze the effect varying $\lambda$ has on our solution for Example 1. In Figure 7, we can see the different paths produced by our method for values of $\lambda \in \{1, 2, 5, 10, 20\}$.




(a) Initial Approach of the Obstacle

(b) Projected Paths

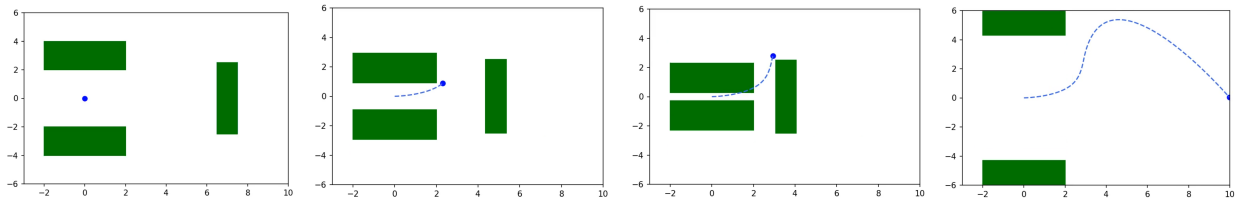
Figure 7: Projected Optimal Paths as $\lambda$ varies for Example 1

Recall from Figure 2, that larger $\lambda$ values correspond to a much sharper cost boundary for the obstacle, while smaller $\lambda$ values correspond to a softer boundary that penalizes the path the closer it is the obstacle.

Given this information, our method does as expected in situations with larger $\lambda$ values and gets very close to the obstacle. Similarly, for smaller values of $\lambda$ our method produces solutions that are very conservative and avoid the obstacle. This is the benefit of having $\lambda$ as a hyperparameter; the user is allowed the flexibility to determine how close the path can get to the obstacle before penalizing. Note that large values of $\lambda$ (greater than 50) often resulted in overflow in our calculations. (see equation (4) and note that derivatives of these functions were taken in (7))

#### 5.3.3 Example 3: Competing Obstacles - How $M$ Influences our Model

Our third example problem was created to see what our method would do to avoid competing obstacles and validate the effect of $M$ on our model. As in Example 1, we started at the point $(0, 0)$, with a target point of $(10, 0)$. We had 3 rectangular obstacles, two of them approaching the starting point from above and below, and a third approaching from the right. The initial setup can be seen in Figure 8(a). We wanted to make sure the obstacles were avoided, so we set $K_i = 500$, and $\lambda_i = 20$ for each obstacle. We also experimented with $M$, the constant that penalizes the magnitude of the acceleration. Below we compare solutions for $M = 8$ and $M = 0.01$.



(a) Initial Setup (b) Avoiding Obstacles 1 & 2 (c) Avoiding Obstacle 3 (d) Full Solution Path

Figure 8: Our Method's Solution for Example 3, M=8

First, setting $M = 8$, we can see in Figure 8(b) our control produces a solution that narrowly misses the first set of colliding obstacles. It also narrowly misses the third obstacle as seen in Figure 8(c). However, the acceleration required to take this path is fairly large, and time is later spent slowing down to change direction and reach the target point.

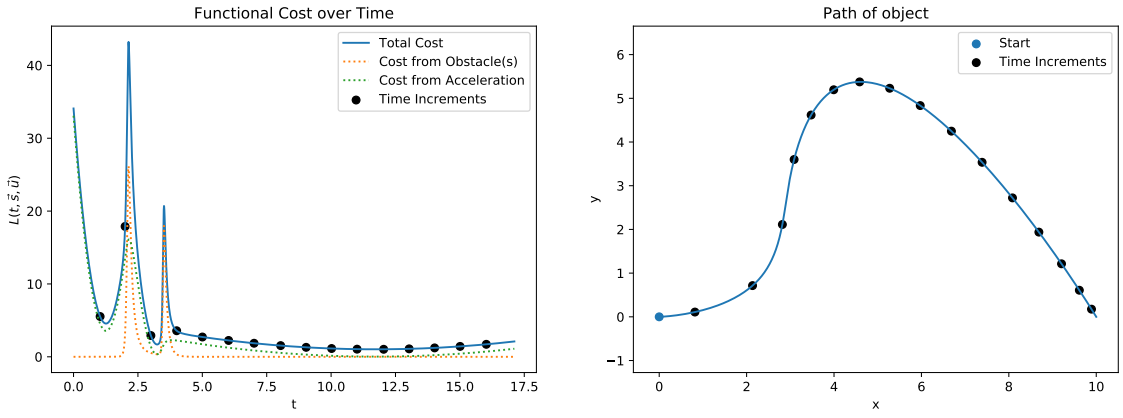
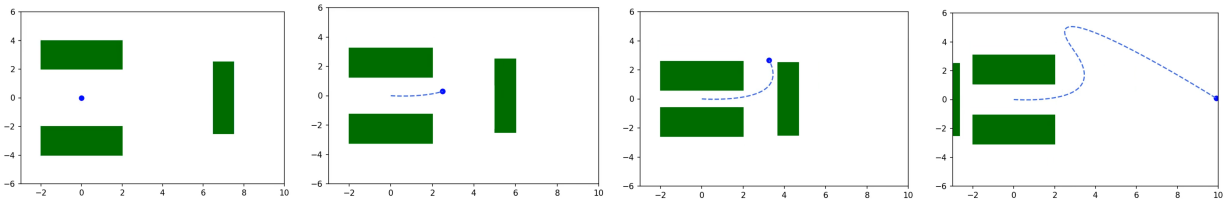



Figure 9: Example 3 - Solution Cost Analysis, M=8

In the cost analysis plot for this solution, (Figure 9), the acceleration's behavior, represented by the sharp peaks in green, matches the description above. We also see peaks for the orange line, which indicate the path got close to each of the obstacles, but still avoided them since otherwise it would have peaked around $K = 500$. With $M = 8$, the total final cost for this solution was 68.311. This is much better than the naive solution which had a total cost of 1241.460.

In order to analyze the effect that the parameter $M$ has on our solution, we changed from $M = 8$ to $M = 0.01$ and reran our method.



(a) Initial Setup (b) Avoiding Obstacles 1 & 2 (c) Avoiding Obstacle 3 (d) Full Solution Path

Figure 10: Our Method's Solution for Example 3, M=0.01

As seen in Figure 10, our method had a similar strategy but the reduced penalty on acceleration allowed our solution to do a much stronger turn to avoid the obstacle on the right. This also allowed for quicker changes in direction to reach the target in almost half the time as before.

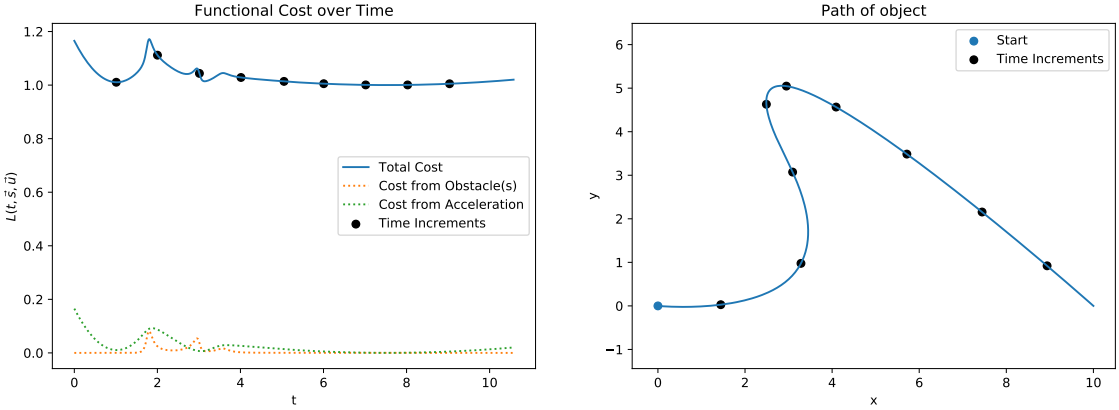



Figure 11: Example 3 - Solution Cost Analysis, M=0.01

The extra mobility of the control allowed by lowering M also led to a much lower overall cost as we can see in the cost analysis plot for our method's solution when $M = 0.01$ (see Figure 11). The final cost for our solution in this scenario was only 10.854, while the final cost for the naive solution under these parameters was 228.084.

From this third example, we found that larger values of $M$ restricted the reaction time of the particle, while smaller values allowed the particle to react more quickly. This indicates that having $M$ as a parameter provides our method with the ability to change the mobility of the particle we are trying to model.

# 6 Conclusion

Overall, our model was successful in finding optimal paths using the maximum principle and Scipy's numerical boundary solver. Model hyperparameters allowed flexibility in path creation by affecting individual obstacle cost, path nearness to obstacles, and particle mobility.

One major drawback to our solution was the requirement of a good initial guess required by scipy.integrate.solve_bvp. Many times, poor initial guesses for the path would result in paths that went through some obstacles or took unrealistic routes to avoid all obstacles. To quote Dr. Benjamin Webb, "Sometimes if you assume have good initial guess, you already assume you know too much."

Many further extensions could be built upon our work. Non-linear paths and varying obstacle shapes would easily fit into our framework. An interesting twist to our problem would be to have objects come into

"view" when they are close enough to the path and have the particle re-adjust its path anytime it "sees" a new obstacle.

# 7 Acknowledgements

We would like to thank Dr. Barker and the entire cohort of BYU ACME professors for their unmatched commitment to our success as students. This project has been an excellent opportunity for us as students to integrate the optimal control theory of this course with our problem-solving skills, mathematical coding, and analysis skills that we have learned in the ACME program. From the bottom of our hearts, we thank you for the countless hours you have all devoted to this program and our futures.

# References

[1] Landry, C., Welz, W. & Gerdts, M. Combining discrete and continuous optimization to solve kinodynamic motion planning problems. Optim Eng 17, 533–556 (2016). https://doi-org.erl.lib.byu.edu/10.1007/s11081-015-9291-0

[2] Har-Peled, S. Approximate Shortest Paths and Geodesic Diameter on a Convex Polytope in Three Dimensions . Discrete Comput Geom 21, 217–231 (1999). https://doi-org.erl.lib.byu.edu/10.1007/PL00009417

[3] https://grapher.mathpix.com

[4] https://www.desmos.com/calculator